THE LAYERED APPROACH TO PREVENT CODE INJECTION ATTACKS

S. Bhuvana
Assistant Professor,
Sriram Engineering College.
bhupreethi@gmail.com

Received 25 January 2018 Received in revised form 04 February 2018 Accepted 06 February 2018

Available online 20 February 2018

ABSTRACT

Code injection attacks, despite being well researched, continue to be a problem today. Modern architectural solutions such as the execute-disable bit and PaX have been useful in limiting the attacks; however, they enforce program layout restrictions and can oftentimes still be circumvented by a determined attacker. We propose a change to the memory architecture of modern processors that addresses the code injection Our experiments with both benchmarks and real-world attacks show the system is effective in preventing a wide range of code injection attacks while incurring reasonable overhead.

Keywords:Code injection attacks,ASLR,PaX

1. INTRODUCTION

The Objective of this system is to foresee code injection attacks which most of the intruders and hackers use to to break into a website. To predict the attack 'split memory technique' is used, it follows the problem at its very root by virtually splitting memory into code memory and data memory.So that a processor will never be able to fetch injected code for execution. This virtual split memory system can be implemented as a software-only patch to an operating system and can be used to supplement existing schemes for improved protection. In addition, our system is able to accommodate a number of response modes when a code injection attack occurs.

Code injection attacks may result in the injection of attack code into the data space. However, the injected code in the data space cannot be fetched for execution as instructions are only retrieved from the code space. Apart from this, address space layout randomization is also followed to prevent code injection attacks. Address space layout randomization deals with storing the contents in the memory randomly instead of storing the entire content in a single memory location. It prevents the attacker obtaining the entire information as the data are stored as patches in different parts of memory.

2. RELATED WORK

The existing system follows von Neumann architecture where the memory cannot split into several segments. This type of technique allows the intruder to inject the code in the single segment and executes it. The intruder takes control of the entire code running in the system and it grants access to modify the data and perform activities without the knowledge of the authorized users. Also address space layout randomization is not possible, the entire data are stored in the single address space, and it allows the third party member to obtain all the valuable information, which is being stored in the database. Also when an attacker modifies a program's data in order to alter program flow, are also not protected by existing system. Even if the system is attacked by an intruder or a third party user, the intruder cannot be tracked.

Drawbacks of Existing System

- Injected code is executed.
- Memory split is not possible.
- Intruder remains invincible.
- URL attacks are not protected.

HARVARD ARCHITECTURE:

The Harvard architecture is computer architecture with physically separate storage and signal pathways for instructions and data. The most obvious characteristic of the Harvard Architecture is

that it has physically separate signals and storage for code and data memory. It is possible to access program memory and data memory simultaneously. Typically, code (or program) memory is read-only and data memory is read-write. Therefore, it is impossible for program contents to be modified by the program itself.

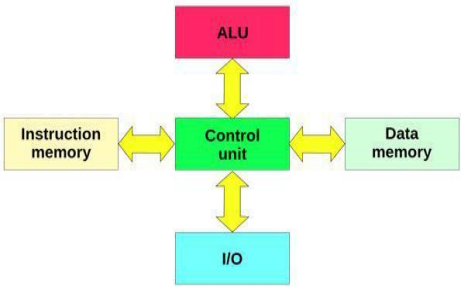

Fig: 2.1 Harved Architecture VON NEUMANN vs. HARVARD

In von Neumann the Same memory holds data, instructions. Also A single set of address/data buses between CPU and memory.

In Harvard separate memories for data and instructions. Also two sets of address/data buses between CPU and memory

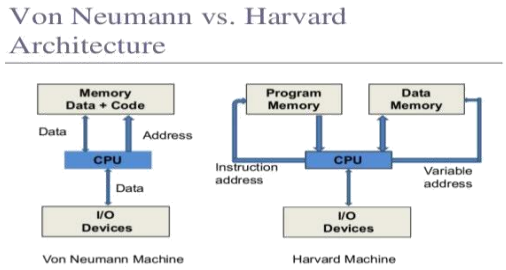

Fig:2.2 Von Neumann vs .Harvard Architecture

3.PROPOSED METHODOLOGY

To forestall the code injection attack, the memory architecture is changed by virtually splitting it into two segments i.e. code segment and data segment. The change in architecture does not allow the intruder to take charge of the injected code, as the injected code remains non-executable. The split memory technique follows Harvard Architecture. Also, Address space layout randomization is followed, where the data are stored in various locations and not as whole in a single memory location. The intruder or an attacker can be tracked by knowing their location, IP address, date and time of the attack etc, that are not available in the existing system.

This section presents how code injection attack is prevented by virtual splitting of architecture. The four phases involved are as:
3.1 Architecture:

Code injection can be prevented with Address space layout randomization phase(ASLR), preventing code injection . Intruders attack by means of URL is prevented by ASLR phase. The intruders can't guess by means of the URL displayed in the Address bar. In the code injection prevention, the system will run code only when it is inserted from the Administrators IP address and host name. Thus intruder's code is not executed and prevented from huge disaster to the website.

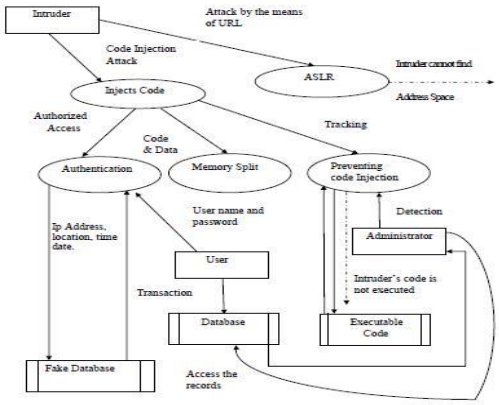

Fig: 3.1.1 Architecture

3.2. Authentication

All applications which contains the user registration and login and administrator's login. In the previous stages, an unknown user also can block the valid user account without knowing the password of the account holder.

3.3. Split Memory

Split Memory phase enables the memory to split into two segments i.e. data and code. The injected code remains in the data segment and it will be unavailable for the processor to execute them. It is based on Harvard Architecture. The control of the system remains in the code segment, as the execution is done only from code segment of the system. The data and code segments are created differently and they are linked with each other through the controls. Data segment allows the user

to enter the details like user name, password etc. Code segment is responsible for the execution of the entire system.

3.3. Address space Intrusion Avoidance Address space layout randomization could be combined with this phase to prevent the URL based attacks. Even if the intruder attacks the system through URL the control will not be granted to the intruder. If the intruder wants to move to next page after the authentication through URL the user remains in the same address, but the page that is being displayed will be different. Also the data are not stored in a single memory space but will be stored in different parts of the memory in patches.

3.4. Code Injection Prevention and Suspension

When the intruder tries to modify any data or create any malicious event, the intruder is not permitted to perform the activities since intrusion is done with unauthorized user name and password. If the changes are done with unauthorized access then the information of the intruder are gathered and it is being sent to the administrator in the secure manner. From those details the intruder can be identified very easily and any further action performed by the intruder can be blocked thus preventing code injection. The details like IP address, hostname, date, time, path, etc., are reported to the system administrator.

4. CONCLUSION

Instead of maintaining single memory space, the memory is split into two as data and code. Even though an intruder injects a code, the code is injected only in the data segment and not in the code segment. Thus the code in the data segment remains unexecuted as the codes are executed only from the code segment. Also the tracking mode enables the administrator to detect the intruder with their specifications. The system is implemented in a security application (like banking security); results show the system is effective in forestalling wide range of code injection attacks

REFERENCES

1. J. Wilander and M. Kamkar, “A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention,” Proc. 10th Network and Distributed System Security Symposium,pp.14962,citeseer.ist.psu.edu/wilander03comparison.html, Feb. 2003.

2.J. von Neumann, “First, Draft of a Report on the EDVAC,” 1945, reprinted in, The Origins of Digital Computers Selected Papers, second ed., pp. 355-364, Springer, 1975.

3.P.C. van Oorschot, A. Somayaji, and G. Wurster,“Hardware-Assisted Circumvention of Self-Hashing Software Tamper Resistance,” IEEE Trans. Dependable and Secure Computing, vol. 2, no. 2, pp. 82-92, Apr. 2005.